ScaMPI User’s guide

Copyright © 1999-2000 Scali AS. All rights reserved.

Acknowledgement

The development of ScaMPI has benefited greatly from the work of people not connected to Scali. We wish especially to thank the developers of MPICH for their contributions to the first ScaMPI implementation.
The list of persons contributing to algorithmic ScaMPI improvements are impossible to compile here. We apologise to those who remain unnamed and mention only those who certainly are responsible for a step forward.

Scali is thankful to Rolf Rabenseifner for the improved reduce algorithm used in ScaMPI.

Table of contents

Chapter 1 Introduction **7**
1.1 Purpose of the ScaMPI User's guide 7
1.2 Scope of the ScaMPI User's guide 7
1.3 Who should read this guide 7
1.4 Acronyms and abbreviations 8
1.5 Terms and conventions 8
1.6 Typographic conventions 9
Chapter 2 Getting started **11**
2.1 The first example 11
2.1.1 Setting up your BASH environment 11
2.1.2 C-source of a Hello-world MPI program 12
2.1.3 Fortran-source of a Hello-world MPI program 12
2.1.4 Compiling 12
2.1.5 Linking 13
2.1.6 Running 13
2.2 MPI test programs 13
2.2.1 A producer-consumer MPI test program 13
2.2.2 A bandwidth MPI test program 14
2.2.3 A bidirectional MPI test program. 14
Chapter 3 Using ScaMPI **15**
3.1 Setting up a ScaMPI environment 15
3.1.1 The Unix ScaMPI environment 16
3.1.2 The Unix on Windows NT ScaMPI environment 17
3.2 Compiling 17
3.2.1 Compiler support 17
3.2.2 Unix compile flags 18
3.2.3 Windows NT compile flags 18
3.3 Linking 19
3.3.1 Linking on Unix 19
3.3.2 Linking on Windows NT 19
3.4 Running MPI programs 20
3.4.1 Mpimon 20
3.4.1.1 Basic usage 20
3.4.1.2 Advanced usage 20
3.5 Debugging ScaMPI applications 25
3.5.1 Debugging with TotalView 25
3.5.2 Debugging with gdb 28
3.6 Profiling ScaMPI applications 29

3.6.1 Profiling with Vampir 29
3.6.2 Profiling with the MPE library 32
Chapter 4 Description of ScaMPI 35
4.1 General description 35
4.1.1 Libraries. 35
4.1.2 Mpimon 35
4.1.3 Mpisubmon 35
4.1.4 Mpiboot 35
4.1.5 Mpid 35
4.2 Startup 35
4.2.1 Phase 1: 36
4.2.2 Phase 2: 37
4.2.3 Phase 3: 38
4.3 Stopping 39
4.4 Communication resources 40
4.4.1 Resources 40
4.4.1.1 Channel 40
4.4.1.2 Eagerbuffer 41
4.4.1.3 Transporter 42
4.4.2 Parameters (mpimon options) 42
4.4.2.1 Prefixes 42
4.4.2.2 Channel 42
4.4.2.3 Eagerbuffer 42
4.4.2.4 Transporter 42
4.4.2.5 Shared memory 43
4.5 Communication protocols 43
4.5.1 Inlining 43
4.5.2 Eagerbuffering 44
4.5.3 Transporter 45
Chapter 5 Getting help 47
5.1 Application notes 47
5.1.1 MPI_Probe and MPI_Recv 47
5.1.2 Unsafe MPI programs 48
5.2 Namespace pollution 48
5.3 Error messages 49
5.3.1 User interface errors 49
5.3.2 Fatal errors 49
5.4 Trouble shooting 50
Chapter 6 Support 55
6.1 Feedback 55
6.2 Problem reports 55
6.3 Platforms 55

Chapter 7 Related Documentation 57
7.1 References 57

Appendix A Install ScaMPI 59
A-1 Installing 59
A-1.1 Requirements 59
A-1.2 Distribution file 59
A-1.3 Licensing 60
A-1.4 Removing an earlier release of ScaMPI 61
A-1.5 Installing a new release 61
A-1.6 Verification of installation 61
A-2 Scali packages file system layout 62
A-3 Useful 3rd party parallel software 62

Chapter 1 Introduction

A Scali system is a cluster of SCI interconnected nodes, where each node is a multiprocessor workstation or PC, running either Solaris, Linux or Windows NT. To get the full computational power of a Scali system it is necessary to use the message passing library ScaMPI. ScaMPI utilises shared memory on intra node communication and the fast SCI interconnect on inter node communication. Any parallel MPI program can be run with ScaMPI and benefit from the SCI performance. This document describes in detail how to use ScaMPI.

1.1 Purpose of the ScaMPI User's guide

This document describes the Scali implementation of MPI (ScaMPI) version 1.1.
Its purpose is twofold:

- to supply the user with enough information to use ScaMPI.
- to give the interested reader an overview of the ScaMPI implementation.

1.2 Scope of the ScaMPI User's guide

This document has the following layout:
Chapter 2 explains how to get started running your first MPI program with ScaMPI.
Chapter 3 describes how to compile, link, run, debug and profile ScaMPI programs.
Chapter 4 describes internal design and functionality of ScaMPI.
Chapter 5 explains what to do if you have problems and gives a list of common errors and solutions.
Chapter 6 explains what to do if you need assistance from Scali.
Chapter 7 gives a list of related documentation you may consult for additional information.
Appendix A explains how to install ScaMPI. The normal user can safely ignore this appendix.

1.3 Who should read this guide

This guide is written for users which have a basic understanding of MPI [1, 2, 3]. It also assumes basic knowledge of 'C'-programming. The guide is *not* a tutorial in using MPI, *nor* a guide how to write efficient MPI programs.

1.4 Acronyms and abbreviations

Abbreviation	Meaning
SCI	Scalable Coherent Interface
MPI	Message Passing Interface
SSP	Scali Software Platform is the generic name of all Scali software packages

Table 1-1: Abbreviations

1.5 Terms and conventions

In all examples default are gcc (GNU C compiler) and bash (GNU Bourne-Again-SHell)

Term	Description.
Node	A single computer in an interconnected system of one or more computers
Cluster	A cluster is a set of interconnected nodes with the possibility to act as a single unit
Scali system	A Scali HPC system = Scali software + compute nodes + high speed interconnect
Process	Instance of application program with unique rank within MPI_COMM_WORLD.
Host	A single node of a Scali system, i.e., a multiprocessor workstation or PC.
Unix	Refers to all UNIX and lookalike OSes supported by the SSP, e.g., Solaris and Linux
WinNT	Refers to Microsoft Windows NT

Table 1-2: Basic terms

1.6 Typographic conventions

Term	Description.
Bold	Program names, options and default values
Italics	User input
#	Command prompt in shell with super user privileges
%	Command promt in shell with normal user privileges
^ > [] ! = ? ""	Notation intended for simple display of Visual GUI setup. ^ is click ^^ is double click ! is uncheck. > is a menu selection chain [] is a window open for multiple selections = is a binary operator, left operand is name or category of selection, right operand is value of selection It is either a predefined value or a specified string " " or ? input according to user context requirements. E.g., File>New>Project[], []=Win32 Console Application, []project name=?, []location=?
	These icons are used througout the text to refer to OS specific details, respectively Solaris and Linux.

Table 1-3: Typographic conventions

Chapter 2 Getting started

This chapter shows how to run your first MPI program in section 2.1. It is also possible to run some test programs that measure basic MPI performance in section 2.2.
All examples are compiled with Gnu or Microsoft Visual compilers and run from a BASH shell. Of course ScaMPI has to be installed and working on your system. If not contact your system administrator or refer to ScaMPI install instructions in appendix A-1. Using other compilers or a Windows NT Visual GUI on these examples might cause some problems. Please refer to the detailed compile, link and run instructions in chapter 3 to overcome any difficulties.

2.1 The first example

Example and test MPI programs are located under /opt/scali/examples src and bin directories.
As a first example you may use the hello-world MPI program in hello.c, or the fortran version hello.f. Complete step by step instructions for a compile, link and run of hello.c and hello.f with ScaMPI are included below.

2.1.1 Setting up your BASH environment

Set MPI_HOME to point to /opt/scali the installation directory of ScaMPI, and set MPI_LDLIBS to include all the required ScaMPI link libraries. Also put the ScaMPI executables in your path.

Unix:

```
% MPI_HOME=/opt/scali; export MPI_HOME
% MPI_LDLIBS="-lmpi; export MPI_LDLIBS
```

WinNT:

```
% MPI_HOME=c:/opt/scali; export MPI_HOME
% MPI_LDLIBS="scampi.lib scabase.lib scacom.lib scasci.lib"; export
MPI_LDLIBS

% PATH=$PATH:$MPI_HOME/bin; export PATH
```

2.1.2 C-source of a Hello-world MPI program

```
#include <stdio.h>
#include “mpi.h”

void main(int argc, char** argv)
{
int rank;
int size;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
printf(“Hello, I'm rank %d; Size is %d\n”, rank, size);
MPI_Finalize();
}
```

2.1.3 Fortran-source of a Hello-world MPI program

```
program hello

implicit none
include 'mpif.h'
integer rank,size,ierr

call mpi_init(ierr)
call mpi_comm_rank(MPI_COMM_WORLD,rank,ierr);
call mpi_comm_size(MPI_COMM_WORLD,size,ierr);
write (*,'(A,I3,A,I3)') "Hello, I'm rank ",rank,'; Size is ',size
call mpi_finalize(ierr);

end
```

2.1.4 Compiling

Unix:

```
% gcc -c -D_REENTRANT -I$MPI_HOME/include hello.c
% g77 -c -D_REENTRANT -I$MPI_HOME/include hello.f
```

WinNT:

```
% cl -c -D_REENTRANT -D_CONSOLE -DWIN32 -DWinNT -MT
-I$MPI_HOME/include hello.c
% f77 -unix -reentrancy:threaded -threads -D_REENTRANT -c -nologo -
DWin32 -DWinNT -iface:cref
-iface:nomixed_str_len_arg -I$MPI_HOME/include hello.f
```

2.1.5 Linking

Unix:
```
% gcc hello.o -L$MPI_HOME/lib -Wl,-R/opt/scali/lib
$MPI_LDLIBS -o hello
% g77 hello.o -L$MPI_HOME/lib -Wl,-R/opt/scali/lib$MPI_LDLIBS -o
hello
```

WinNT:
```
%cl -nologo -MT hello.obj -o hello
-link -LIBPATH:$MPI_HOME/lib $MPI_LDLIBS
%f77 -unix -nologo -reentrancy:threaded -threads hello.obj -
Fehello.exe -link -LIBPATH:$MPI_HOME/lib
```

2.1.6 Running

Run the hello world example on 3 Unix nodes named hostA, hostB, hostC or a Windows NT node with name hostN.

Unix:
```
% mpimon hello -- hostA 1 hostB 1 hostC 1
```
WinNT:
```
% mpimon hello.exe -- hostN 3
```

The hello example will produce the following output:

```
Hello, I'm rank 0; Size is 3
Hello, I'm rank 2; Size is 3
Hello, I'm rank 1; Size is 3
```

2.2 MPI test programs

The ScaMPItst package contains some MPI test programs. If ScaMPItst is installed on your system you have MPI test sources and binaries under the /opt/scali/examples src and bin directories.
These programs are given a short presentation here together with simple run instructions.
It is of course possible to use the included makefiles to compile the MPI test sources. No details are mentioned, since the compile, link and run process are already described in the first example hello-world program.

2.2.1 A producer-consumer MPI test program

Producer is a simple producer-consumer program. Processes with rank 0, 1, 2, ..., n/2-1 send data while process n/2, n/2+1, ...,n-1 receive data. Process 0 will send to process n-1 and process 1 will send to process n-2 and so on.
The *producer* program parameters are:

-l *i* ***i*** is the loop count,

-n *j* ***j*** is the number of bytes to transfer for each send operation.
As a first test, run *producer* between any pair of two nodes, nodeX and nodeY:

```
% mpimon producer -l 1 -n 1024 -- nodeX nodeY
```

A single process is started on each node and a single message of size 1024 bytes are transferred from the process on nodeX to the process on nodeY. The program should return "TEST COMPLETE".

Repeat the test for all pairs of nodes. N is the number of hosts (must be a even number for this test).
The program should return "TEST COMPLETE".

```
% mpimon producer -l 1 -n 1024 -- <node1> <node2> ...<nodeN>
```

2.2.2 A bandwidth MPI test program

Bandwidth is a program to measure bandwidth for various messages sizes between two processes. First one way bandwidth and the latency for a zero byte message are measured, then the ping-pong (two way) bandwidth and latency.
Measure the bandwidth between any pair of nodes, nodeX and nodeY, by running:

```
% mpimon bandwidth -- nodeX nodeY
```

2.2.3 A bidirectional MPI test program.

Bidirect tests uni- and bi-directional traffic between a given number of nodes. The program may be run between two nodes, nodeX and nodeY, as:

```
% run_bidirect nodeX nodeY
```

or between a set of given nodes, nodeX nodeY ... nodeZ , as:

```
% run_permutated_bidirect nodeX nodeY ... nodeZ
```

The run_permutated_bidirect script will test uni- and bi-directional traffic between all permutations of node combinations.

Chapter 3 Using ScaMPI

This chapter describes the setup, compile, link, run, debug and profile of an MPI program using ScaMPI. The control and start up of any MPI program using ScaMPI is monitored by *mpimon*.
Running ScaMPI on a Scali cluster with multi-user and resource management software, implies that the startup mechanism is built on top of *mpimon*. Only mpimon is described in this document. We advocate use of state-of-the-art cluster management software, it is a user friendly and effective way of utilising a Scali cluster. See the documentation at your computer lab about cluster management.

Contents summary:
Section 3.1.1 explains how to setup the ScaMPI environment in a BASH shell for Unix or Unix on Windows NT. Compile and link instructions are given respectively in section 3.2 and 3.3. How to run a ScaMPI program with *mpimon* is explained in section 3.4, which contains a list of all *mpimon* program parameters. Debugging ScaMPI programs is explained in section 3.5 and performance profiling of ScaMPI programs is explained in section 3.6.

3.1 Setting up a ScaMPI environment

System administration propably has set up your ScaMPI shell environment in startup scripts. Environment variables point to the ScaMPI installation directory , the ScaMPI executables must be located in your path and ScaMPI libraries and dynamic link library paths has to be defined. If you have a proper ScaMPI environment setup you may safely skip section 3.1.

3.1.1 The Unix ScaMPI environment

The use of ScaMPI requires that some environment-variables are defined. These are usually set in the standard startup scrips (e.g. .bashrc when using BASH, gnu Borne Again SHell), but they can also be defined manually:

Name	Description
MPI_HOME	Installation directory. For a standard installation this should be set as `export MPI_HOME=/opt/scali`
MPI_LDLIBS	Library to be loaded, defined as ld-directives. For a standard installation the necessary libraries are: `export MPI_LDLIBS="$CRT_BEGIN -lmpi $CRT_END"` CRT_BEGIN and CRT_END should be empty if no value is given for the compiler. Details can be found in the release notes.

LD_LIBRARY_PATH	The LD_LIBRARY_PATH variable may be updated to include the directory where the dynamic libraries can be found.
	`export LD_LIBRARY_PATH=`
	`${LD_LIBRARY_PATH}:$MPI_HOME/lib`
	An alternative to use LD_LIBRARY_PATH is to give a flag to the linker which can include a specific path to the dynamic libraries, see the release notes.

PATH	The PATH variable may be updated to include the directory where the mpi binaries can be found. `export PATH=${PATH}:$MPI_HOME/bin`

Table 3-1: Environment variables on Unix

3.1.2 The Unix on Windows NT ScaMPI environment

At present there are no Unix on Windows NT (e.g., Cygnus, Interix) available with MT safe POSIX interface, therefore ScaMPI on Windows NT only supports Windows MPI programs. It is however possible to run ScaMPI from a Cygnus BASH shell and use a Unix makefile system with the Visual C++ and Fortran compilers. A few environment variables must be set:

Name	Description
MPI_HOME	Installation directory. For a standard installation this should be set as `MPI_HOME=c:/opt/scali; export MPI_HOME`
MPI_LDLIBS	Library to be loaded, defined as ld-directives. For a standard installation the necessary libraries are: `export MPI_LDLIBS="scampi.lib scacom.lib scabase.lib scasci.lib"` Details can be found in the release notes.

LD_LIBRARY_PATH	The LD_LIBRARY_PATH variable may be updated to include the directory where the dynamic libraries can be found.
	`export LD_LIBRARY_PATH=`
	`${LD_LIBRARY_PATH}:$MPI_HOME/lib`
	An alternative to use LD_LIBRARY_PATH is to give a flag to the linker which can include a specific path to the dynamic libraries, see the release notes.

PATH	The PATH variable may be updated to include the directory where the mpi binaries can be found. `export PATH=${PATH}:$MPI_HOME/bin`

Table 3-2: Environment variables for Unix on Windows NT

3.2 Compiling

ScaMPI is an API (Application Programming Interface) and not an ABI (Application Binary Interface), hence all applications must be recompiled and linked with ScaMPI.

3.2.1 Compiler support

Unix: ScaMPI is a C++ library built using GNU g++, hence the MPI libraries must be linked in differently depending on the users choice of compilers. Check the release notes for details regarding support of your compiler. Note: the ScaFgcc package (or a similar version of gcc) must be installed on the system.

Solaris: ScaMPI is supported for use with the listed compilers:
- GNU and EGCS gcc/g++/g77 for UltraSPARC and i86pc

- Apogee apcc/apCC/apf77/apf90 for UltraSPARC
- Portland Group pgcc/pgf77/pgf90 i86pc
- Sun SunPro - CC/f77/f90 for UltraSPARC and i86pc

WinNT: ScaMPI is supported for use with following compilers:
- MS Visual C++/Digital Visual Fortran for i86pc.

3.2.2 Unix compile flags

The following string must be included as compile flags (BASH syntax):

```
"-D_REENTRANT -I$MPI_HOME/include"
```

3.2.3 Windows NT compile flags

WinNT: Building a ScaMPI application on Windows NT using a GUI is described in detail, since this approach is very different from an ordinary compile session on unix. Please check the release notes if you need additional information.

Microsoft Visual C++/Digital Fortran
1) Start MS Visual C++ or Digital Visual Fortran

2) Choose a project name and a project location. It is recommended that the program location is at a shared file system available from all the nodes where the program shall be run.
File>New>Project[],[]=*Win32 Console Application*,[]project name=?, []location=?

3) Add c, c++ or fortran files to the project.
Project>Add To Project>Files=?

Settings for compiling c or c++ code: (skip if only fortran source)
4) Add a preprocessor definition and location of additional include files.
Project>Settings[], []Settings For=*All Configurations*,[] C/C++>Preprocessor [] , []Preprocessor Definitions=*WinNT*, []Additional include directories=*/opt/scali/include*.

5) Select a multithreaded runtime library for debug or release code.
Project>Settings[], [] Settings For=*Win32 Debug*, [] C/C++>Code Generation[]Use Runtime Library=*Debug Multithreaded*, or
Project>Settings[], [] Settings For=*Win32 Release*, []C/C++>Code Generation[]Use Runtime Library=*Multithreaded*.

Compiling Fortran code: (skip if only c/c++ source)

6) Add a preprocessor definition and include path.
Project>Settings[], []Settings For=*All Configurations*, [] Fortran>Preprocessor [] , [Preprocessor Definitions=*WinNT*, []Custom INCLUDE=*c:/opt/scali/include*, []Use Path=*c:/opt/scali/include*.

7) Select multithreaded runtime libraries, enable reentrancy support.
Project>Settings[], [] Settings For=*All Configurations*, []Fortran>Libraries [] , []=*Use Multi-threaded Library*,[]=*Enable Reentrancy Support*

8)Choose default external procedures calling interface.
Project>Settings[], []Settings For=All Configurations, [] Fortran>External Procedures[] , []Default Calling=*C, By Reference*.

3.3 Linking

3.3.1 Linking on Unix

The following string give the setup for necessary link flags (BASH syntax):

```
"-L/opt/scali/lib -Wl,-R/opt/scali/lib $CRT_BEGIN -lmpi $CRT_END"
```

Since it is required to link with the GNU runtime library, the syntax is dependent on the compiler you use. Please, check the release notes. The runtime setup CRT_BEGIN and CRT_END libraries are defined for some compilers. -R is a flag asking the linker to include the path to the dynamic libraries, so that the environment variable LD_LIBRARY_PATH is not needed, see table 3-1. When linking a Fortran main program include the Fortran interface library -lfmpi before CRT_BEGIN.

3.3.2 Linking on Windows NT

Linking on a Windows NT GUI is explained in detail, and continues the description of compiling on WinNT, section 3.2.2. Check the release notes if you need additional information.

Linking (both for fortran and c/c++ source)
9) Add ScaMPI library path and ScaMPI libraries.
Project>Settings[], [] Settings For=*All Configurations*, [] Link>Input [], []Object/library modules=*"ScaMPI.lib ScaBase.lib ScaCom.lib ScaSci.lib"*, []Additional library path=*"c:/opt/scali/lib"*.

10) It is recommended to enable the "startup banner" for compiling and linking:
Project>Settings[], [] Settings For=*All Configurations*, [] Link>Customize [] , []!Suppress startup banner.

Project>Settings[], []Settings For=*All Configuration*s, [] C/C++>Customize [] , []!Suppress startup banner.
You should now be able to build your project.

3.4 Running MPI programs

Executables containing ScaMPI-calls cannot be started directly from a shell prompt. They *must* be started using **mpimon**; the control-/startup-program supplied with a ScaMPI-distribution.
Remember it is absolutely necessary to run an ScaMPI program with a homogene file system image, providing the same path and program name on all nodes of the cluster. Mpimon can be started from a shell prompt or the Scali Desktop GUI [4]. It is also possible that ScaMPI programs are started from a 3rd party software workload management system on top of mpimon [6, 16].

3.4.1 Mpimon

mpimon have many options which can be used for optimising ScaMPI-performance. Normally it should not be necessary to use any of these. However unsafe MPI programs [3] might need buffer adjustments to get rid of hangs (it is of course not a recommended method of programming). Trading performance by changing communication space is best avoided if there are no compelling reason to do so and you know what to do.

3.4.1.1 Basic usage

mpimon <user program> <program options> -- <hostname> [<count>] [<hostname> [<count>]]...
will be normal use.

Option	Description
<userprogram>	Name of application program
<programoptions>	Program options to application program.
--	Separator, signals end of user program options.
[<hostname> [<count>]]	Pair of name of host and number of processes to run on that host. Hosts can occur several times in the list. Processes will be given ranks sequentially according the list of host-number-pairs.

Table 3-3: Basic options to mpimon

3.4.1.2 Advanced usage

mpimon [<mpimon-option>]... <program & host-spec> [-- <program & host-spec>]...

is the complete syntax for using **mpimon.**

Parameter	Description
<program & host-spec>	**<program spec> -- <host spec> [<host spec>] ...**
<program spec>	**<userprogram>[<programoptions>]...**
<userprogram>	Name of application program
<programoptions>	Program options to application program.
--	Separator, signals end of user program options.
<host spec>	**<hostname> [<count>]**
<hostname>	Name of host.
<count>	Number of processes to run on host. **mpimon** will start 1 process on host if **<count>** is omitted.

Table 3-4: Mpimon parameters

Numeric values can be given as mpimon options in the following way:

Option	Description
<numeric value>	**<decimal value> | <decimal value><postfix>**
<postfix>	**<K>: <numeric value> = <decimal value> * 1024** **<M>: <numeric value> = <decimal value> * 1024 * 1024**

Table 3-5: Numeric input

Mpimon option	Description
-automatic <*selection*>	Set automatic-mode for process(es). Default: **none** Legal: **'n,m,o..' = (list) or 'n-m' = (range) or 'all'**
-barrier_fanin <*count*>	Set number of barrier fanin reads. Default: **8**
-barrier_fanout <*count*>	Set number of barrier fanin reads. Default: **8**
-debug <*selection*>	Set debug-mode for process(es). Default: **none** Legal: **'n,m,o..' = (list) or 'n-m' = (range) or 'all'**
-debugger <*debugger*>	Set debugger to start in debug-mode.
-disable-timeout	Disable process timeout
-display <*display*>	Set display to use in debug-/manual-mode.
-exact_match	Set exact-match-mode.
-execpath <*execpath*>	Set path to internal executables.
-environment <*value*>	Define how to export environment. Default: **none** Legal: **'export' = all or 'mpi' = MPI_?? or 'none'**
-help	Display this.
-home <*directory*>	Set installation-directory.
-immediate_handling <*selection*>	Handling of immediates Default: **lazy** Legal: **lazy, threaded, automatic**
-inherit_limits	Inherit userdefinable limits to processes
-init_comm_world	Init COMM_WORLD at startup(all channels are created)
-inter_adapters <*adapters*>	Set list of sci adapters for inter-communication. Default: **all** Legal: **'n,m,o..' = (list) or 'n-m' = (range) or 'all'**
-inter_channel_inline_threshold <*size*>	Set threshold for inlining (in bytes) per inter-channel. Default: **560**

Table 3-6: Complete list of mpimon options

Mpimon option	Description
-inter_channel_size <*size*>	Set buffer size (in bytes) per inter-channel. Default: **4K** Legal: **Powers of 2**
-inter_chunk_size <*size*>	Set chunk-size for inter-communication. Default: **512k** Legal: **Multiplum of pages**
-inter_eager_count <*count*>	Set number of buffers for eager inter-protocol. Default: **2**
-inter_eager_size <*size*>	Set buffer size (in bytes) for eager inter-protocol. Default: **128K** Legal: **Powers of 2**
-inter_pool_size <*size*>	Set buffer-pool-size for inter-communication. Default: **32M** Legal: **Multiplum of pages**
-inter_transporter_count <*count*>	Set number of buffers for transporter inter-protocol. Default: **4** Legal: **Powers of 2**
-inter_transporter_size <*size*>	Set buffer size (in bytes) for transporter inter-protocol. Default: **64K**
-intra_channel_inline_threshold <*size*>	Set threshold for inlining (in bytes) per intra-channel. Default: **560**
-intra_channel_size <*size*>	Set buffer size (in bytes) per intra-channel. Default: **4K** Legal: **Powers of 2**
-intra_chunk_size <*size*>	Set chunk-size for intra-communication. Default: **1M** Legal: **Multiplum of pages**
-intra_eager_count <*count*>	Set number of buffers for eager intra-protocol. Default: **2**
-intra_eager_size <*size*>	Set buffer size (in bytes) for eager intra-protocol. Default: **128K** Legal: **Powers of 2**
-intra_pool_size <*size*>	Set buffer-pool-size for intra-communication. Default: **4M** Legal: **Multiplum of pages**

Table 3-6: Complete list of mpimon options

Mpimon option	Description
-intra_transporter_count <*count*>	Set number of buffers for transporter intra-protocol. Default: **4** Legal: **Powers of 2**
-intra_transporter_size <*size*>	Set buffer size (in bytes) for transporter intra-protocol. Default: **64K**
-manual <*selection*>	Set manual-mode for process(es). Default: **none** Legal: **'n,m,o..' = (list) or 'n-m' = (range) or 'all'**
-separate_output <*selection*>	Enable separate output for process(es).Filename: ScaMPIoutput_host_pid_rank Default: **none** Legal: **'n,m,o..' = (list) or 'n-m' = (range) or 'all'**
-sm_debug <*selection*>	Set debug-mode for submonitor(s) Default: **none** Legal: **'n,m,o..' = (list) or 'n-m' = (range) or 'all'**
-sm_manual <*selection*>	Set manual-mode for submonitor(s) Default: **none** Legal: **'n,m,o..' = (list) or 'n-m' = (range) or 'all'**
-sm_trace <*selection*>	Enable trace for submonitor(es) Default: **none** Legal: **'n,m,o..' = (list) or 'n-m' = (range) or 'all'**
-statistics	Enable statistics.
-stdin <*selection*>	Distribute standard in to process(es) Default: **none** Legal: **'n,m,o..' = (list) or 'n-m' = (range) or 'all'**
-verbose	Display values for user-options.
-timeout <*timeout*>	Set timeout(elapsed time i seconds) for run Legal:

Table 3-6: Complete list of mpimon options

3.5 Debugging ScaMPI applications

Debugging ScaMPI applications is recommended using a parallel debugger. Manual debugging with a separate debugging session for each MPI process requires no parallel debugger. It may however become a time consuming and tedious task when debugging manually several processes. Currently the only parallel debugger option is TotalView, with support for Solaris and a subset of compilers. Support for other OSes and compilers can be made available, contact **sales@scali.com** for further information. Starting a ScaMPI program with a parallel debugger, you can stop the program in MPI_Init. Here the parallel debugger give you a single point of control for the debugged MPI program(s).
A manual debug session is started with mpimon options e.g. as **-debugger** *dbx* **-debug** *0*. An xterm window is started for each of the specified MPI processes, in this example process *0*. A message containing the program run parameters needed by the debugger is displayed. Then the debugger, here *dbx*, is started. The user must manually insert the run parameters to the debugger for all debug processes, before setting breakpoints or any other debugging actions.

3.5.1 Debugging with TotalView

This section describes TotalView briefly, explains how to set up the user environment and gives the first getting started with ScaMPI and TotalView instructions. Refer to the TotalView User's Guide documentation for TotalView specific explanations.

What is TotalView?
TotalView is the recommended parallel debugger on a Scali system running ScaMPI.
• The TotalView graphical parallel debugger has support for the most important parallel programming abstractions, threads, MPI, PVM and HPF.
• TotalView has a fast and easy to learn GUI, with the possibility of data visualisation, a useful aid in debugging numerical programs.

• TotalView is designed for multiprocessing and offer the debugger features a programmer expects.

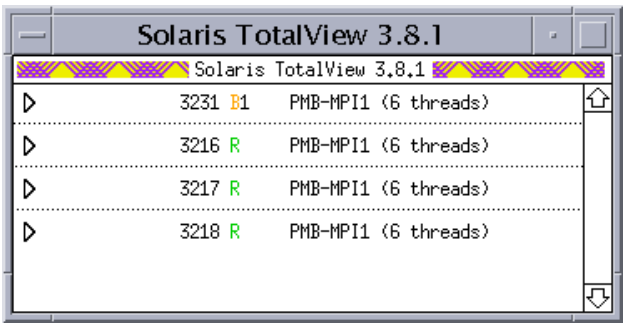
Solaris TotalView 3.8.1
Solaris TotalView 3.8.1
3231 B1 PMB-MPI1 (6 threads)
3216 R PMB-MPI1 (6 threads)
3217 R PMB-MPI1 (6 threads)
3218 R PMB-MPI1 (6 threads)

Figure 3-1: TotalView main window

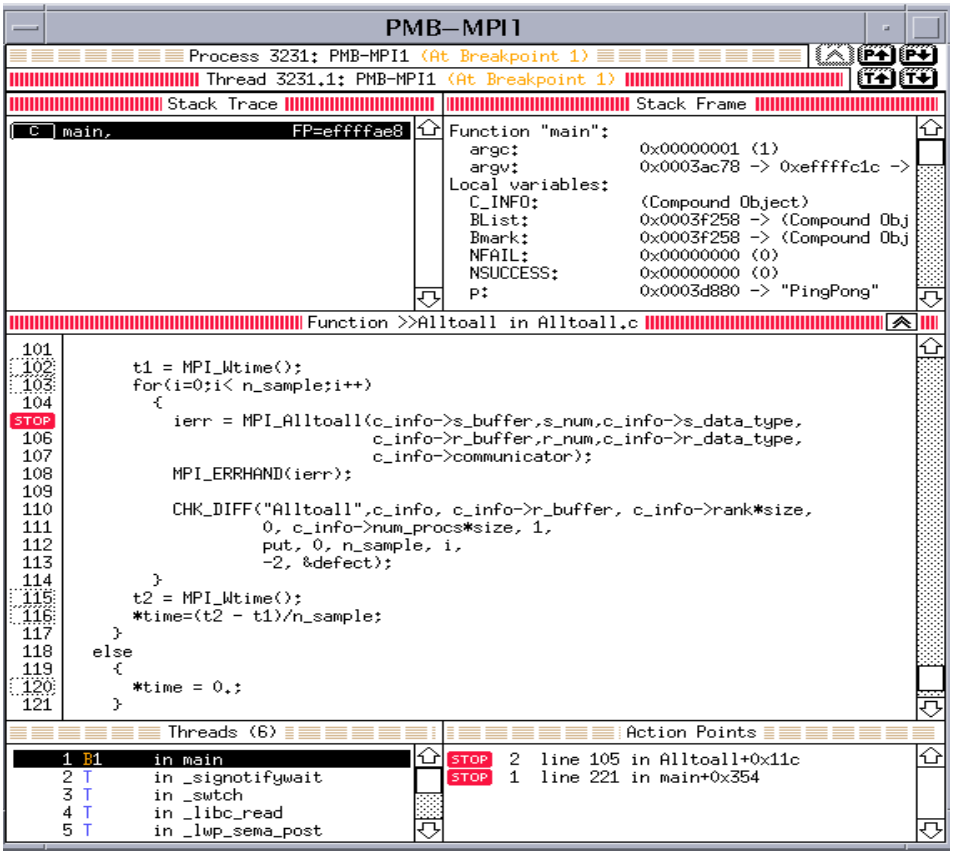

Figure 3-2: TotalView MPI process window

TotalView user environment setup

TotalView users must have a few environment variables defined, see table 3-7, giving the locations of binaries, man pages, libraries and licenses. This environment should be set automatically by your system administration. If you need more information check the TotalView Installation Guide [8].

Name	Description
PATH	points to directory where the TotalView binaries are installed. e.g. /opt/totalview/bin
MANPATH	points to directory where the TotalView man pages are installed e.g. /opt/totalview/man
LD_LIBRARY_PATH	points to directory where the TotalView libraries are placed, e.g. /opt/totalview/lib
LM_LICENSE_FILE	a ":" separated list which points to the TotalView license file, e.g.: /opt/totalview/license.dat

Table 3-7: Environment setup for TotalView

Run a ScaMPI program from within the TotalView debugger
First of all compile your programs with the -g flag to include debug symbols in the executables. Depending on your compilier it may be necessary to specify what kind of debug symbols to be generated, see the ScaMPI release notes. Use

```
tvmpimon <mpimon options>
```

to start an MPI program with ScaMPI in a single TotalView debugger window. See the TotalView User's Guide [7], for help on debugging with TotalView. TotalView ask if you want to stop in MPI_Init. A stop give you a single point of control of the debugging session. Use the middle mouse button to get a menu of possible commands. The point and click interface of TotalView should get you easily started.

Current TotalView support
At the moment TotalView is only supported on Solaris. The message queue feature of TotalView with MPI is not yet implemented, and there may be some restrictions on debugging threads. See the TotalView release notes and the ScaMPI release notes for further details.

3.5.2 Debugging with gdb

The default *mpimon* **-debugger** is the GNU debugger gdb. Specify the processes you want to debug with the **-debug** option to *mpimon*. Cut the xterm displayed run parameters and paste them into gdb when you want to start the debugged process.

3.6 Profiling ScaMPI applications

For MPI programs the most useful tool beside a parallel debugger is a MPI performance tracing tool showing the message passing performance of a program run. There are two MPI profiling options with ScaMPI. Recommended is VAMPIR a commercially supported library, but it is possible to use MPE, a freely available MPI profiling library (available from Scali as the ScaMPE package).
The performance trace libraries are linked with ScaMPI libraries through a profiling interface defined in the MPI standard. Performance data is output to a file when the instrumented program reaches MPI_Finalize. The VAMPIR software is currently only supported with ScaMPI on Solaris, but can be made available on other platforms. Contact **sales@scali.com** for more information.

3.6.1 Profiling with Vampir

This section contains a brief description of VAMPIR, how to setup the user environment and basic instructions about how to collect performance data and how to analyse the performance.
Refer to the VAMPIR User's Guide documentation for VAMPIR specific explanations.

What is VAMPIR?

- Vampir MPI performance analysis help you organise the performance data, understand application and communication behaviour, evaluate load balancing and identify communication hots pots.
- A timeline window display application and message passing activities and shows parallelism as the sum of participating processes. Communication statistics can be displayed for selected intervals of time and message length.
- Execution statistics for selected call trees can be displayed, and comparisons made on data from multiple runs. Other features include source code display and animation.

A few of the common global information displays are shown in the following figure 3-3 and 3-4.

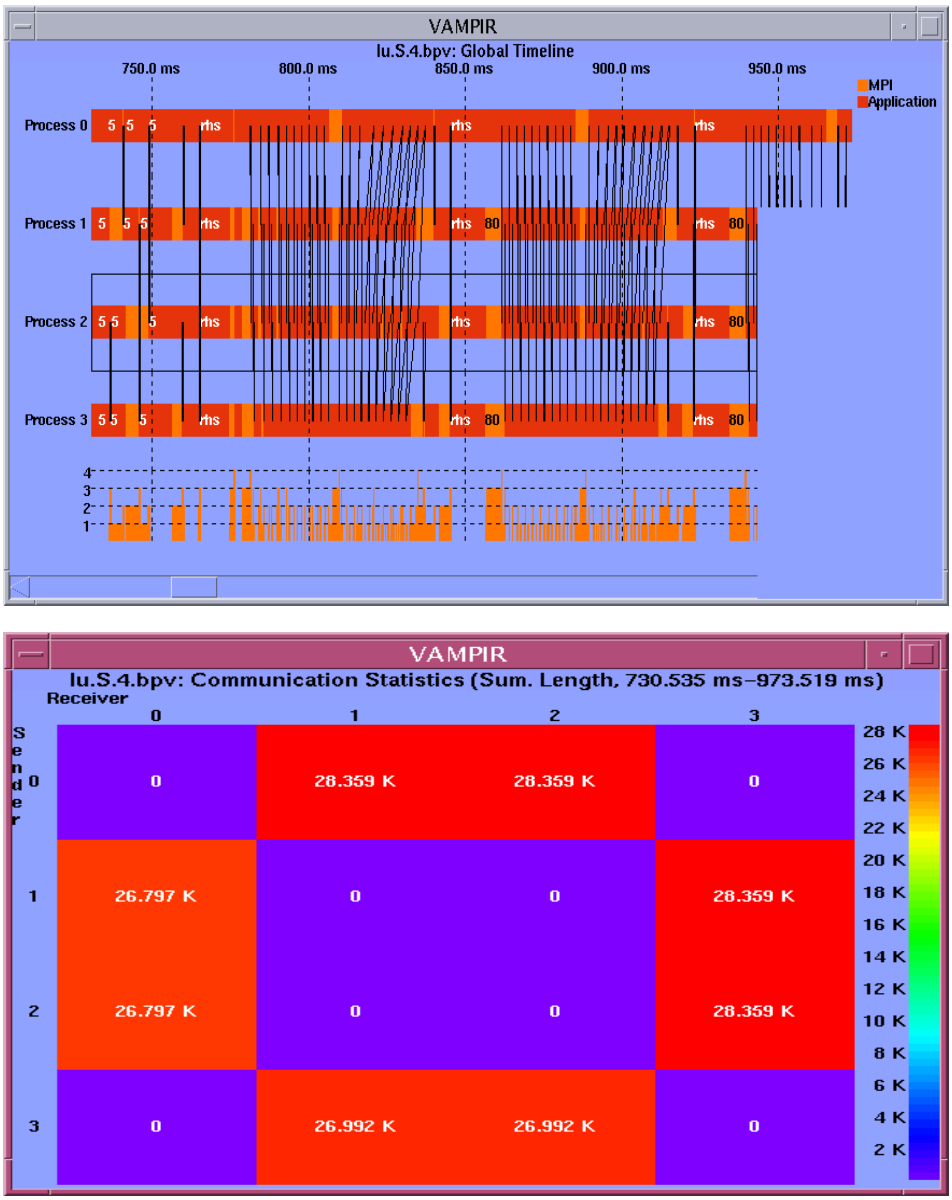

Figure 3-3: VAMPIR snapshots

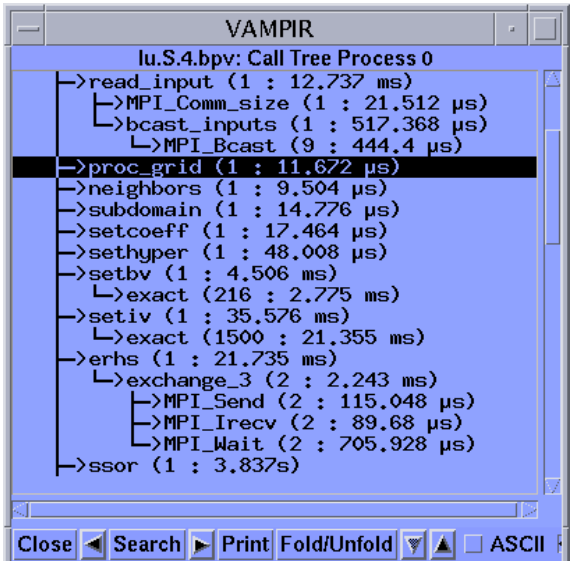
VAMPIR
lu.S.4.bpv: Call Tree Process 0
->read_input (1 : 12.737 ms)
->MPI_Comm_size (1 : 21.512 µs)
->bcast_inputs (1 : 517.368 µs)
->MPI_Bcast (9 : 444.4 µs)
->proc_grid (1 : 11.672 µs)
->neighbors (1 : 9.504 µs)
->subdomain (1 : 14.776 µs)
->setcoeff (1 : 17.464 µs)
->sethyper (1 : 48.008 µs)
->setbv (1 : 4.506 ms)
->exact (216 : 2.775 ms)
->setiv (1 : 35.576 ms)
->exact (1500 : 21.355 ms)
->erhs (1 : 21.735 ms)
->exchange_3 (2 : 2.243 ms)
->MPI_Send (2 : 115.048 µs)
->MPI_Irecv (2 : 89.68 µs)
->MPI_Wait (2 : 705.928 µs)
->ssor (1 : 3.837s)
Close
Search
Print
Fold/Unfold
ASCII

Figure 3-4: VAMPIR profiled MPI program call tree window

VAMPIR user environment setup

To use VAMPIRtrace, a user must have access to the VAMPIRtrace directories, and the environment variables for the license scheme must be set correctly. Most likely the VAMPIR environment is setup automatically by your system administration. If not, the license keys are stored in a plain ASCII file, the pathname of which must be made known to VAMPIRtrace by setting one of the two environment variables in table 3-8.

Name	Description
PAL_ROOT	points to the root of the VAMPIRtrace installation
PAL_LICENSEFILE	specifies the complete pathname of the license key file

Table 3-8: Setup for VAMPIR and VAMPIRtrace

Create a VAMPIR trace file

Link your MPI program with VAMPIRtrace by adding the VT library before your ScaMPI libraries on the link line (BASH syntax):

```
-L$VAMPIR_LIB_DIR -lfmpi -lVT -lmpi ...
```

This example shows the placement of the VAMPIRtrace library for a fortran program, remove the reference to the fortran interface library libfmpi when linking a C main program.
Run the program with *mpimon* **-environment export** to collect performance data. It is important to export the environment so that all participating processes have an appropriate license setup. When MPI communication stops, the trace is written from memory to a file and a message is sent to stdout .
It is also possible to instrument a program manually with VAMPIRtrace calls. The extra overhead caused by profiling code can be turned off by linking with the dummy library -lVTnull instead of the real -lVT library. The DIMEMAS performance prediction library can also be substituted for the performance analysis VAMPIR library, contact **sales@scali.com** if you want this option.

Analyse the program performance with VAMPIR
Start the VAMPIR analyse program e.g.,

```
% vampir &
```

Use *File™Open Tracefile* and browse and select the appropriate trace file for analysis. The point and click interface of VAMPIR should get you easily started. Figure 3-3 and 3-4 show some global information displays offered by VAMPIR.
Look in the VAMPIRtrace Installation and User's Guide on ScaMPI [7] for further information.

3.6.2 Profiling with the MPE library

ScaMPE is a modified version of the freely available MPE (Multi Processing Environment) libraries from MPICH. An executable linked with the ScaMPE library collect performance data during runtime with output to a file.

The main components of the MPE library are :
- A set of routines for creating logfiles for examination by e.g, jumpshot, upshot or nupshot.
- Trace or real time animation of MPI calls.
- A shared-display parallel X graphics library.

To link with the MPE libraries include one of the following libraries before the -lmpi library.

- -ltmpi

Trace all MPI calls. Each MPI call is preceded by a line that contains the rank in MPI_COMM_WORLD of the calling process, and followed by another line indicating that the call has completed. Most send and receive routines also indicate the values of count, tag, and partner (destination for sends, source for receives). Output is to standard output.

- -llmpi -lm

Generate an upshot-style log file of all MPI calls. The name of the output file is executablename_profile.log. For example, if the program is sendrecv, the generated log file
is sendrecv_profile.log.

- -lampi -lmpe -lm -lX11

Produces a real-time animation of the program. This requires the MPE graphics, and uses X11 Window System operations. You may need to provide a specific path for the X11 libraries
(instead of -lX11).
In Fortran, it is necessary to include the library **-lfmpi** ahead of the profiling libraries (a part of ScaMPI). This allows C routines to be used for implementing the profiling libraries
for use by both C and Fortran programs. For example, to generate files in a Fortran program, the library list is -lfmpi -llmpi -lm.

Analyse the program performance with Upshot
Start the upshot or any other analyse program that understands the clog format e.g., nupshot or jumpshot:

```
% upshot &
```

Browse and select the appropriate trace file for analysis. MPICH documentation [14] includes a description of the MPE graphics routines.
The ScaMPE package is downloadable from http://www.scali.com.

Chapter 4 Description of ScaMPI

4.1 General description

ScaMPI consists of libraries to be loaded with the user application program and a set of executables which control the startup and execution of the user application program.

4.1.1 Libraries.

Name	Description
libmpi	Standard library containing 'C' api.
libfmpi	Library containing 'fortran' api wrappers.

Table 4-1: Libraries

4.1.2 Mpimon

Mpimon is a monitor program which is the user interface for running the application program.

4.1.3 Mpisubmon

Mpisubmon is a submonitor program which controls execution of application program. One mpisubmon is started on each host per run.

4.1.4 Mpiboot

Mpiboot is a bootstrap program used when running in manual-/debug-mode.

4.1.5 Mpid

Mpid is a daemon program running on all hosts that can run ScaMPI. Mpid is used for starting mpisubmon programs to avoid using UNIX facilities like remote shell which has proven to be error prone. Mpid is started automatically when a host boots and must run at all times.

4.2 Startup

ScaMPI uses sockets for control purposes. Schematically a startup of a mpi cluster is done as follows

4.2.1 Phase 1:

Step	Description
Parameter control	**mpimon** does as much control of options and parameters as possible, user-program names are checked to be ok and hosts are contacted through sockets to see if they are alive and that **mpid** is running.
Connecting to mpids	**mpimon** connects to daemons on all hosts and transfers basic information to enable **mpid** to start submonitor.

Table 4-2: Startup phase 1

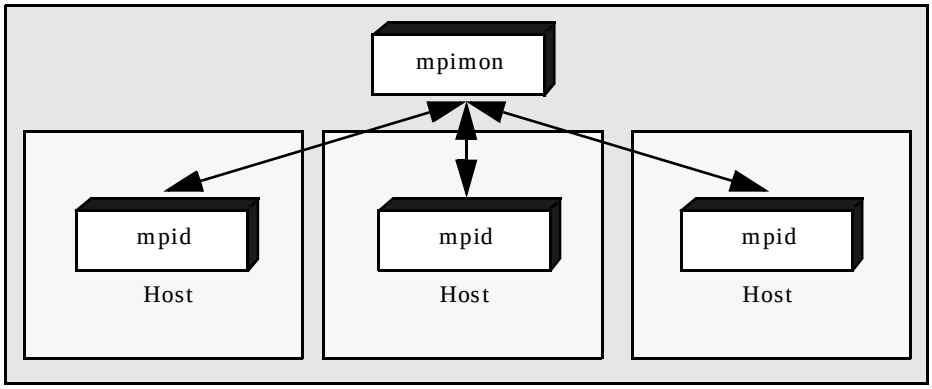

Figure 4-1: Startup phase 1

4.2.2 Phase 2:

Step	Description
Submonitors starts	**mpisubmon** is started by **mpid** and connects to mpimon for a control socket, information to enable **mpisubmon** to start userprograms(processes) are transferred. **mpisubmon** creates shared memory areas.

Table 4-3: Startup phase 2

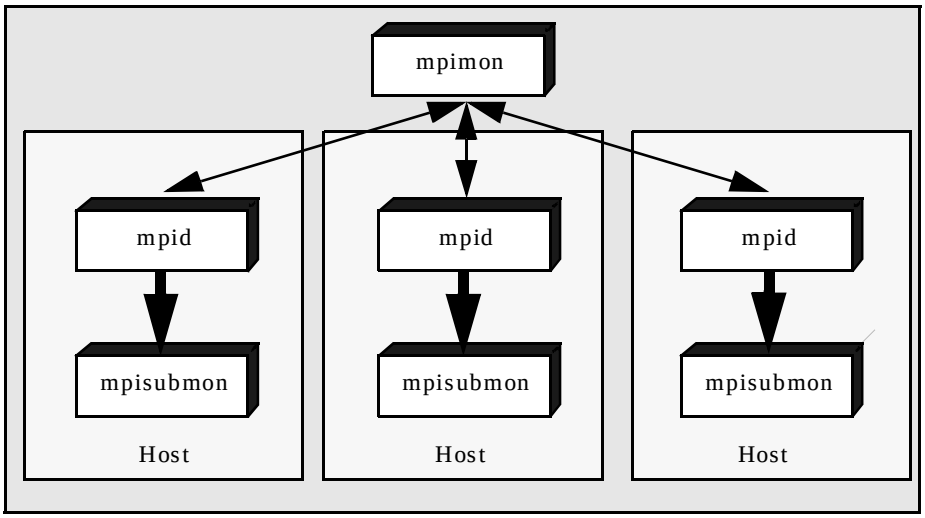

Figure 4-2: Startup phase 2

4.2.3 Phase 3:

Step	Description
Processes start and enter MPI_Init	**mpisubmon** starts all processes to be run on the host where it is running.
Processes synchronize	When the processes have received all control information they will signal **mpimon** that they are ready to run. **mpimon** will then send a start running message to all processes.
Processes return from MPI_Init and start running.	User program takes control.

Table 4-4: Startup phase 3

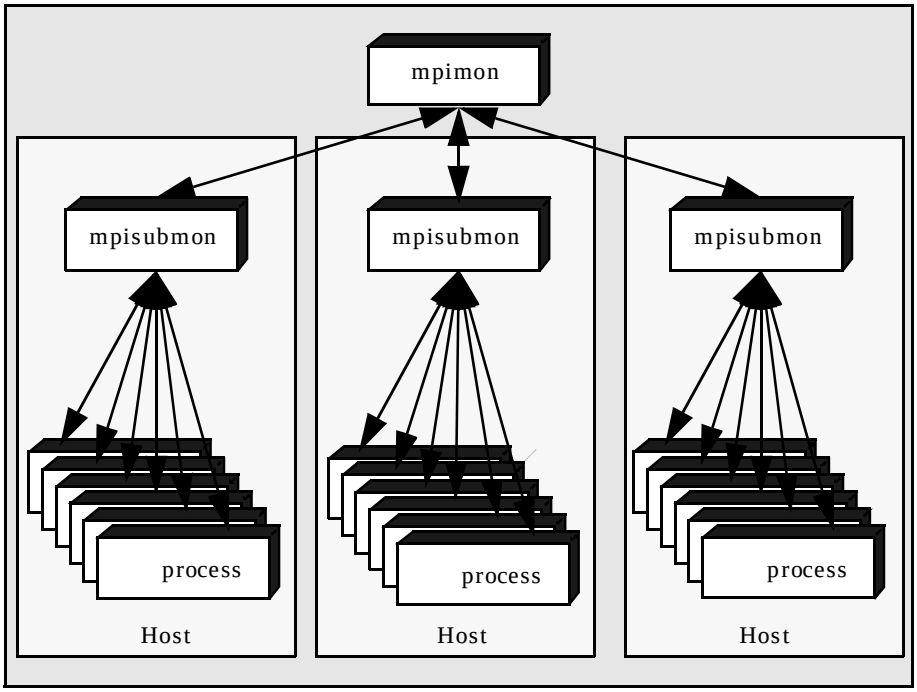

Figure 4-3: Startup phase 3

4.3 Stopping

Step	Description
Processes enter MPI_Finalize	Processes signal to mpimon that they have entered finalize.
Processes synchronize	Processes wait for an “all stopped message” from mpimon (sent when all processes are in finalize).
Processes leave MPI_Finalize	Processes terminate and submonitors release shared memory segments and exit, mpimon terminates.

Table 4-5: Stopping steps.

4.4 Communication resources

ScaMPI uses a “on demand” scheme for allocating resources. This means that resources will be allocated when needed. All resources reside in shared memory and are allocated by **mpisubmon** on demand from the sender process.

4.4.1 Resources

4.4.1.1 Channel

A channel is a unidirectional connection between a sender and a receiver. There will be one channel per sender-/receiver-pair for each communicator. Each ringbuffer entry is 64-byte and contains the message envelope..

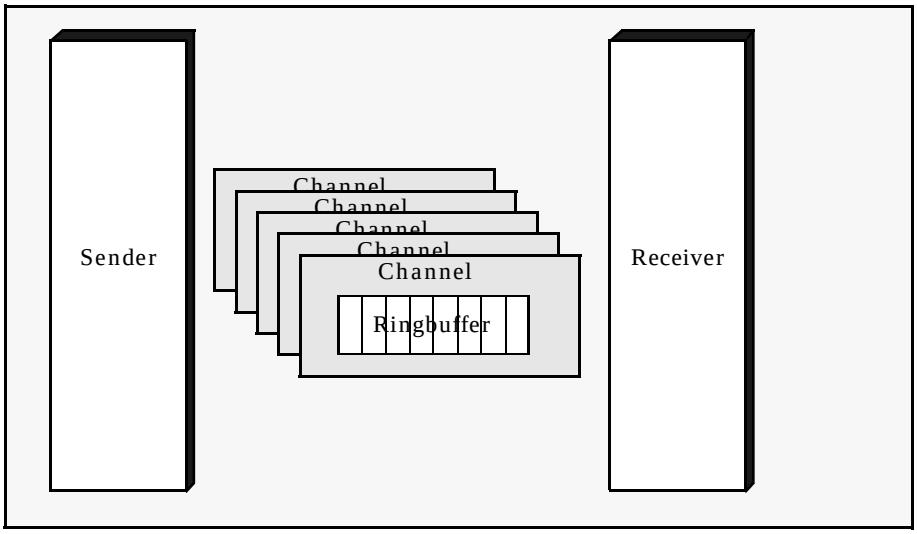

Figure 4-4: Channel

4.4.1.2 Eagerbuffer

Eagerbuffer is a buffered connection between a sender and a receiver. There will be only one eagerbuffer for a sender-/receiver-pair.

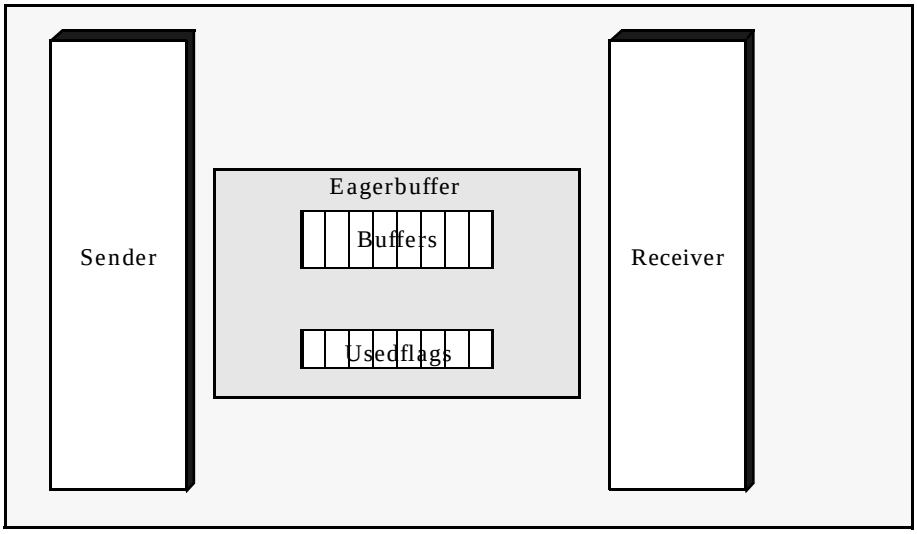

Figure 4-5: Eagerbuffer

4.4.1.3 Transporter

Transporter is a connection between a sender and a receiver for large datatransfers. There will be only one transporter for a sender-/receiver-pair.

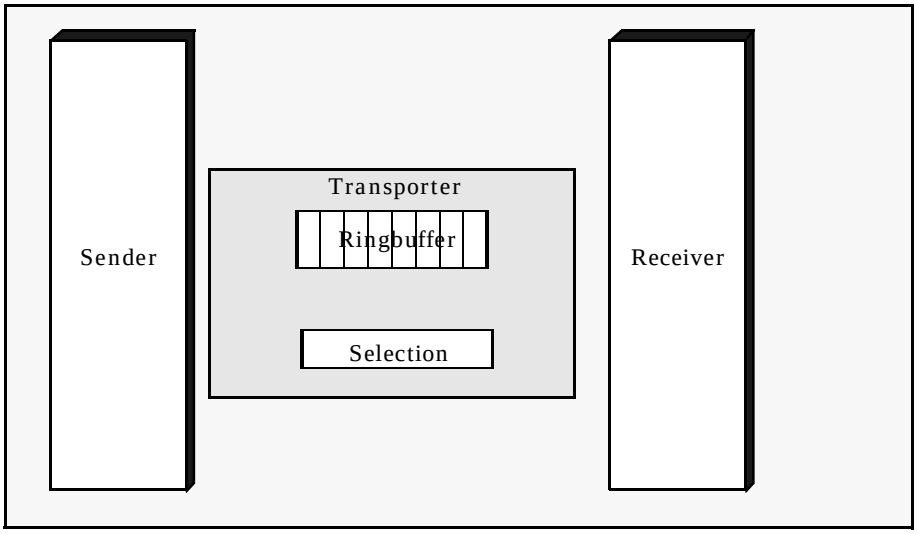

Figure 4-6: Transporter

4.4.2 Parameters (mpimon options)

4.4.2.1 Prefixes

The parameters concerning resources have a prefix which is either “-inter” meaning between two hosts or “-intra” meaning within same host.

4.4.2.2 Channel

The size of channel ringbuffer can be set with “-inter_/-intra_channel_size”.

4.4.2.3 Eagerbuffer

The size of each eagerbuffer can be set with “-inter_/-intra_eager_size”.
The number of eagerbuffers can be set with “-inter_/-intra_eager_count”.

4.4.2.4 Transporter

The size of each ringbufferpart can be set with “-inter_/-intra_transporter_size”.
The number of ringbufferparts can be set with “-inter_/-intra_transporter_count”.

4.4.2.5 Shared memory

mpisubmon has two pools of shared memory, one for inter-host resources and the other for intra-host resources. The size of each chunk and the number of chunks in each pool can be set by options “-inter_/-intra_pool_size” and “-inter/-intra_pool_count”.

4.5 Communication protocols

ScaMPI employs different protocols for communication depending on the size of the message.

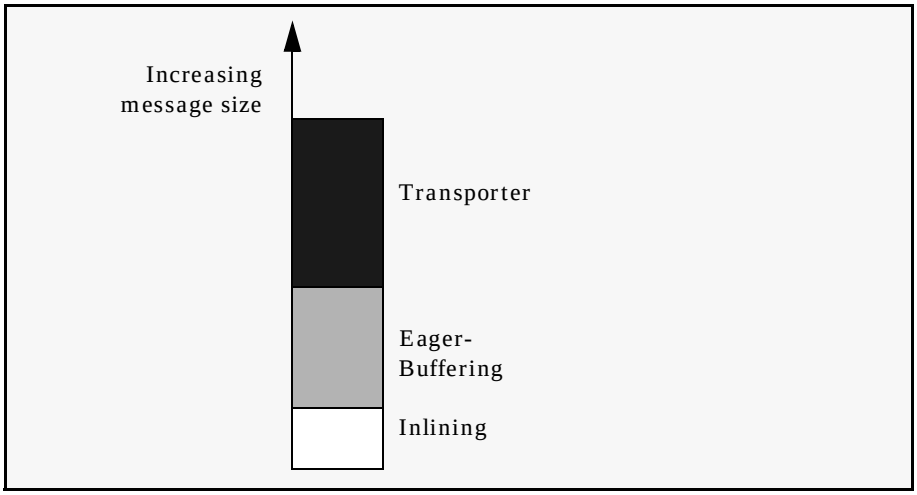

Figure 4-7: Thresholds for different communication protocols

4.5.1 Inlining

Inlining means including data in the header and is used for small messages(< inter_/intra_channel_inlining_theshold bytes).

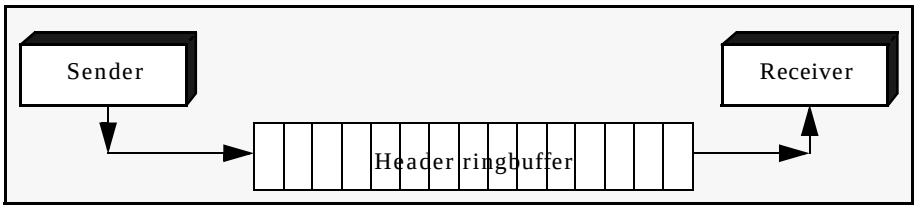

Figure 4-8: Inlining

4.5.2 Eagerbuffering

Eagerbuffering uses a scheme where buffers are allocated by the sender of a message and are released by the receiver without any explicit communication between the two.

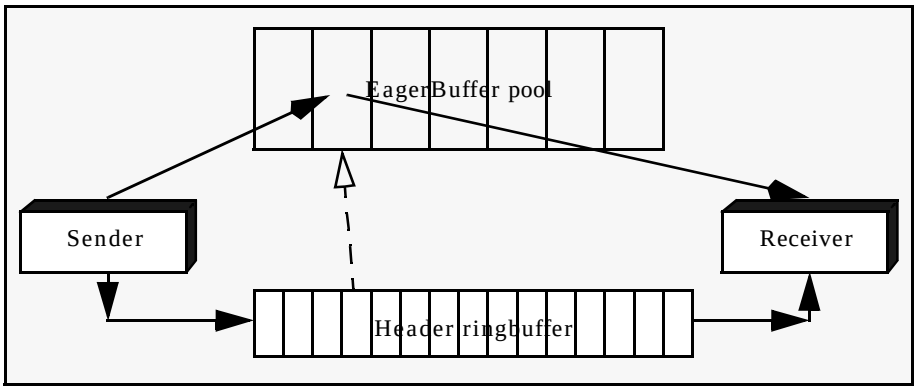

Figure 4-9: Eagerbuffering

4.5.3 Transporter

For large messages we use a “rendez-vous” communication protocol called transporter. The sender only sends a header (phase 1)and when the receiver is ready to accept the data it communicates this to the sender (phase 2). Data is then transferred using dedicated buffers (phase 3).

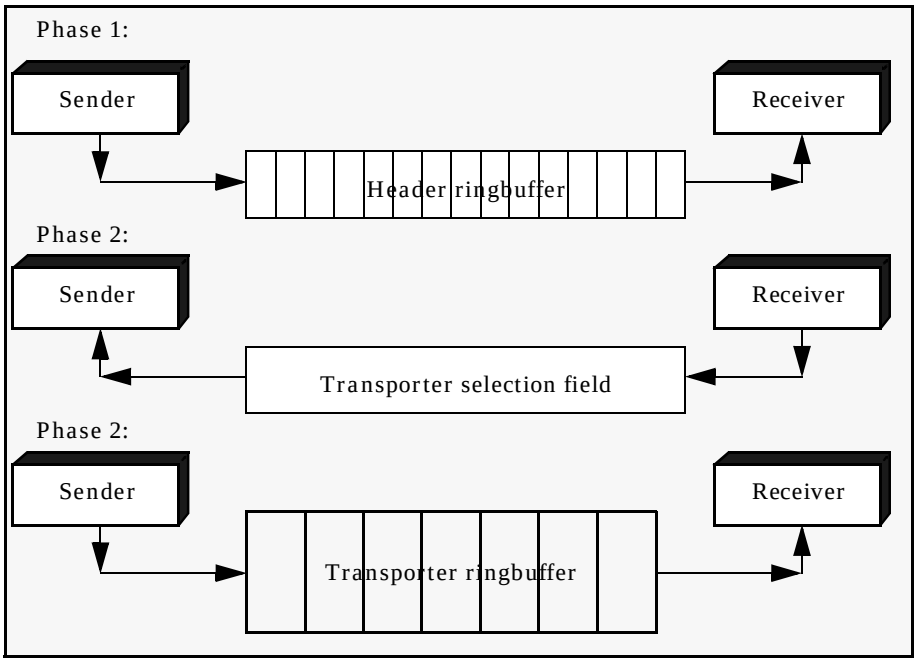

Figure 4-10: Transporter

Chapter 5 Getting help

This chapter is the place to start when something goes wrong running your ScaMPI programs. If you have any problems with ScaMPI, first check the not yet complete list of common errors and their solutions. Eventually a compiled and updated list will be available on a frequently asked questions (FAQ) page at http://www.scali.com. If you cannot get help about how to solve your problems here, please get assistance from Scali. Read the support chapter 6 first before you contact **support@scali.com**.

Contents summary:
Section 5.1 contains some application notes about why some programs run with MPICH and not ScaMPI. Section 5.2 give a list of all the names to avoid when programming with ScaMPI. Section 5.3 contains error messages given by ScaMPI, and section 5.4 about trouble shooting give the solution to common problems. These sections are by no means complete now, but problems reported to Scali will eventually get into this ScaMPI trouble shooting chapter. So please send your relevant complaints to Scali.

5.1 Application notes

5.1.1 MPI_Probe and MPI_Recv

During development and test of ScaMPI we have run into several application programs with the following code sequence:

```
{
.
while (...) {
MPI_Probe(MPI_ANY_SOURCE, MPI_ANY_TAG, comm, sts);
if (sts->MPI_TAG == SOME_VALUE) {
      MPI_Recv(buf, cnt, dtype, MPI_ANY_SOURCE,
                  MPI_ANY_TAG, comm, sts);
      doStuff();
}
.
}
}
```

This sequence works ok for implementations that have one receive-queue for all senders. ScaMPI have one receive-queue per sender. This gives messages from different senders the possibility to bypass each other. The problem with the sequence

above is that a sent message could arrive in the timegap between the probe finishes and the receives matches a message; i.e it is not certain that the message that probe matched is the same as the one that recv matches. The correct sequence should be:

```
{
        .
        while (...) {
        MPI_Probe(MPI_ANY_SOURCE, MPI_ANY_TAG, comm, sts);
        if (sts->MPI_TAG == SOME_VALUE) {
              MPI_Recv(buf, cnt, dtype, sts->MPI_SOURCE,
                          sts->MPI_TAG, comm, sts);
              doStuff();
        }
        .
        }
        }
```

5.1.2 Unsafe MPI programs

Some programs may run with MPICH and not ScaMPI because of different buffering behaviour. Unsafe MPI programs may require resources that are not always guaranteed by ScaMPI and dead lock. If you want to know more about how to write portable MPI programs, see e.g. [2].

5.2 Namespace pollution

The ScaMPI library is written in C++ and we have prefixed all our classes with “MPI_”. Depending on the compiler used, the user may run into problems if he/she has C++ code with prefix “MPI_”. We have also a few global variables that could cause problems.

Name	Description
MPI_log	Pointer to log-/error-facility-structure
mpipriv	Common block for all “global variables” when linking with Fortran code. (defined in mpif.h)
PMPI_CommIdent	C routine which returns a unique identifier for each communicator, used in trace-routines.
PMPI_PrintAllState	C routine which can be used to print some extra state, used when debugging/ optimization.

Table 5-1: Namespace pollution

Name	Description
PMPI_PrintCommState	C routine which can be used to print some extra state, used when debugging/ optimization.
PMPI_PrintRecvChannelState	C routine which can be used to print some extra state, used when debugging/ optimization.
PMPI_PrintSendChannelState	C routine which can be used to print some extra state, used when debugging/ optimization.

Table 5-1: Namespace pollution

5.3 Error messages

5.3.1 User interface errors

User interface errors are problems with the environment setup causing difficulties for mpimon when starting a ScaMPI program. Mpimon will not start before the environment is properly defined. These problems are usually easy to fix, by giving mpimon the correct location of some executable. The error message give a stright forward indication of what to do. Only particularly troublesome user interface errors will be listed here.

5.3.2 Fatal errors

At a fatal error ScaMPI writes an error message before starting MPI_Abort to shut down all MPI processes.

5.4 Trouble shooting

Problem	*Description* / Solution
How do I start my program? *use mpimon from shell prompt ...* *use mpimon from Scali Desktop GUI ...*	See Getting started in chapter 2 for practical examples. If your program is reading from stdin remember to use the mpimon option **-stdin 0** to send the input to at least one MPI process. Default is **-stdin none**. It is also possible to use the Scali Desktop GUI: % /opt/scali/bin/scadesktop &
Why doesn't my program start to run (simple)? *set the application and library paths ...*	*mpimon: command not found.* FIX: Include /opt/scali/bin in you PATH environment variable *mpimon can't find mpisubmon.* FIX: Set MPI_HOME=/opt/scali or use the -execpath option. *The application has problems loading libraries (libsca*).* FIX-1: Update the LD_LIBRARY_PATH to include /opt/scali/lib. FIX-2: Link your application with path to Scali libraries (gnu: -Wl,-R/opt/scali/lib).
Why doesn't my program start to run (advanced)? *interconnect problems: routing ...*	*The program hangs in MPI_Init() and terminate in ICMS_NO_RESPONSE.* CHECK: Check if routing is properly set with scaconftool. It is easiest to use the Scali Desktop GUI.
Why doesn't my program start to run (advanced)? *interconnect problems: bad clean up ...*	*A previous ScaMPI run has not been properly terminated.* CHECK: Use e.g., /opt/scali/bin/scaps. *A process holds SCI or shared memory resources, (Core dumping takes time ...).* CHECK-1: Use /opt/scali/bin/scidle CHECK-2: Use /opt/scali/bin/scish to check for SHM segments (ipcs for Solaris & Linux, [TBD] for WinNT)
Why doesn't my program start to run (advanced)? *interconnect problems: space overflow ...*	*Your application have required too much SCI or shared memory resources.* CHECK: Your mpimon size specifications are too large, check with: mpimon -verbose ... CHECK: Number of communicators in the program is higher than expected.
Why doesn't my program start to run (advanced)? *core dump ...*	*The application core dumps.* CHECK: Use a parallel debugger e.g., TotalView to find the point of violation after an appliacation recompile for inclusion of debug symbol information, (-g for most compilers). A manual debug session is possible with gdb, dbx, pgdbg or any available sequential debugger.

Table 5-2: Trouble shooting

Problem	*Description* / Solution
Why does mpid not start?	*mpid opens a socket with a fixed identification (obtained from getservbyname with name mpid). If mpid is terminated abnormally then this socket will not be available before a timeout.* Use "netstat -a | grep mpid" to observe when socket is released and start mpid again.
How to resolve incompatible mpi-versions?	*mpid, mpimon, mpisubmon and the libraries all have version variables that are tested at startup.* An incompatibility have normally one of two reasons: A new version of ScaMPI is installed without restart of mpid or the environment variable 'MPI_HOME" is set wrongly.
Why does my application terminate abnormally?	*Are you reasonable certain that your algorithms are MPI safe?* *The program just hangs:* Try to start your program with -init_comm_world specified - if it doesn't start you have a buffer allocation problem (read controlling shared memory allocation). If you have large degree of asynchronicity try to increase the channel_size else are you really sure that your algorithms are MPI safe? *The program terminate without with error message:* If core was dumped look at the file else try again with -verbose.
Why does my application terminate abnormally? ***SCI interconnect failures ...***	*The program terminate with ICMS_FAILURE.* *The program terminate with ICMS_OUTBOUND_MAP_FULL.* This is a SCI problem and a reload of SCI drivers may be neccessary. Check the SCI documentation in the Scali system guide and your system administration about the cause of the problem. Use the interconnect diagnostic in the Scali Desktop GUI. Contact **support@scali.com** if there is a SCI problem needing attention. Problems and fixes will be included in a FAQ on http://www.scali/com.
What monitor-options are available?	Run mpimon without any options to get the list of all mpimon options with a short explanation. Or check the mpimon description in chapter 4.
How do I control SCI and local shared memory usage? (simple) *adjusting ScaMPI buffer sizes ...*	Forcing size parameters to mpimon is usually not neccessary. This is only a means of optimising ScaMPI to a particular application on knowledge of communication patterns. For unsafe MPI programs it may be required to adjust buffering to allow the program complete.

Table 5-2: Trouble shooting

Problem	*Description* / Solution
How do I control SCI and local shared memory usage? (advanced) *one communciation channel ...*	*The eager buffers are used for small messages, while the transporter buffers are used for handling large messages (larger than eager size).* *The function of the various buffers is detailed in the "ScaMPI user's guide"* *All buffers are created at start up when -init_comm_world is specified or when needed (tried used for the first time). The channel buffers is a send queue where each entry is 64 bytes i.e. there is room for 64 outstanding requests in a 4k buffer. The bufferspace required by a communication channel is approximately:* channel = (2 * channel_size * communicators) + (transporter_size * transporter_count) + (eager_size * eager_count) + 512
How do I control SCI and local shared memory usage? (advanced) *pool size ...*	*The communicators parameter is dependent of the application (assumed to be two in the automatic approach). If more communicators than expected by the buffer size calculations are used, the application may run out of shared memory. The pool size is a limit for the total amount of shared memory. Default pool size is set to 32M inter and 4M intra node.*
How do I control SCI and local shared memory usage? (advanced) *automatic buffer management ...*	*The automatic buffer size computations is based on a full connectivity, i.e., all communicating with all others. If all process communicate with all others, they will communicate P_inter processes inter node and with P_intra processes intra node (one-self inclusive). Each communication channel is therefore restricted to use only:* inter_part = inter_pool_size / (P_inter*P_intra) intra_part = intra_pool_size / (P_inter*P_intra*P_intra) *The automatic approach is to downsize all buffers associated with a communication channel until it fits in its part of the pool. The chunk size sets the size of each individual allocated memory segment. The automatic chunk size is calculated to wrap a complete communication channel.*
How do I control SCI and local shared memory usage? (advanced) *barrier buffer ...*	The barrier buffer is one page and up to barrier_fanout+1 (default 8+1) buffermappings are created.

Table 5-2: Trouble shooting

Problem	*Description* / Solution
How do I control SCI and local shared memory usage? (advanced) *an example ... with ScaMPI 1.6.4:*	*Running two processes on one node with channel-size 256k* mpimon -intra_channel_size 256k -intra_pool_size 4m \ /opt/scali/examples/bin/bandwidth -- <nodename> 2 *Would terminate without starting with the message:* --- uiError --- Intra_pool_size must be at least 4227072 bytes (2113536 bytes * 2 processes) for given set of parameters . (Calculation is left out as an exercise ...). *Using the minimum pool size:* mpimon -intra_channel_size 256k -intra_pool_size 4227072 \ /opt/scali/examples/bin/bandwidth -- <nodename> 2 *Would start with the following parameters :* -intra_channel_size 256K -intra_chunk_size 1M -intra_eager_count 1 -intra_eager_size 1K -intra_pool_size 4227072 -intra_transporter_count 4 -intra_transporter_size 512 A more natural choice of parameters may be: mpimon -intra_channel_size 256k -intra_pool_size 6m \ /opt/scali/examples/bin/bandwidth -- <nodename> 2 Would start with the following parameters : -intra_channel_size 256K -intra_chunk_size 1M -intra_eager_count 2 -intra_eager_size 64K -intra_pool_size 6m -intra_transporter_count 4 -intra_transporter_size 64K Note: channel_size 256k is an unusual high value.
How do I optimise MPI performance? *perfomance analysis ...*	Learn about the performance behaviour of your MPI application on a Scali system by using a performance analysis tool. Recommended are the VAMPIR MPI profiling software for ScaMPI, contact sales@support.com for further details. It is also possible to use the freely available ScaMPE profiling library with ScaMPI.
How do I optimise MPI performance? *uneffective isend,irecv ...*	I*f communication and calculations does not overlap using MPI_Isend, MPI_Irecv or variants of these are usually performance ineffective.*

Table 5-2: Trouble shooting

Problem	*Description* / Solution
How do I optimise MPI performance? *avoid starving processes - fairness ...*	*MPI programs may, if not special care is taken, be unfair and may starve processes ,e.g., by using MPI_Waitany() as illustrated for a client-server application in example 3.15 & 3.16 in the MPI 1.1 standard.* Fairness can be enforced, e.g., by use of several tags or separate communicators.
How do I optimise MPI performance? *avoid using isend,irecv creating many threads beyond number of processors ...*	*When immediate send are used ScaMPI creates an additional thread for handling this. The same applies for immediate receive, so a total of two extra threads may be created when using immediate function calls. Having more than one thread on a multi-processor usually improve performance, while increasing the number of threads beyond the number of processors may reduce performance (due to active waiting and context switch time).*
How do I optimise MPI performance? *using sendrecv transforms to isend / irecv ...*	*In ScaMPI MPI_Sendrecv transforms into MPI_Isend & MPI_Recv i.e. two threads. Using MPI_Sendrecv to communicate on two processes on one node then transform into 4 threads (assuming application is single threaded) and may be slow on a dual processor node.* For small messages (<= eager_size) MPI_Isend is treated as MPI_Send and hence acceptable performance. To shift the performance degradation to out of observation field set the -intra_eager_size 1M (or a size larger than the largest message size).

Table 5-2: Trouble shooting

Chapter 6 Support

6.1 Feedback

We welcome any suggestions, improvements, feedback or bug reports on this User Guide and the software described herein. Please send your comments to **support@scali.com.**
We also encourage the user of parallel tools software with ScaMPI on Scali clusters to comment any aspects of the software to the NHSE, National HPCC Software Exchange branch - Parallel Tools Library [12]. It is a site open to the public giving review of parallel tools and allowing users, software vendors and others to respond and express their views and frustrations. Please help us improve the parallel software on our Scali clusters!

Other important mail groups are **scali-announce** and **scali-user**. Join the appropriate mailing group on **http://www.scali.com**.

6.2 Problem reports

Problem reports should include software version, computer architecture, an example and a record of the sequence of events causing the problem.

6.3 Platforms

ScaMPI is available on Scali SCI clustered Intel PC's with either Solaris, Linux or Windows NT. ScaMPI is available on Scali SCI clustered Sun UltraSPARC workstations with Solaris.
For further information contact **sales@scali.com** or visit Scali's website **http://www.scali.com**.

Chapter 7 Related Documentation

7.1 References

[1] **"MPI: A Message-Passing Interface Standard"**, The Message-Passing Interface Forum, Version 1.1, June 12, 1995. *http://www.mpi-forum.org*.

[2] **"MPI: The complete Reference: Volume 1, The MPI Core",** Marc Snir, Steve W. Otto, Steven Huss-Lederman, David W. Walker, Jack Dongarra. 2e, 1998. The MIT Press.

[3] **"MPI: The complete Reference: Volume 2, The MPI Extension"**, William Grop, Steven Huss-Lederman, Ewing Lusk, Bill Nitzberg, William Saphir, Marc Snir. 1998. The MIT Press.

[4] **"Scali System Guide"**. Scali AS, *http://www.scali.com.*

[5] **"ScaMPI release notes".** Scali AS, *http://www.scali.com.*

[6] **"CCS: Computing Center Software resource management for networked high-performance computers"**. Paderborn Center of Parallel Computing, *http://www.uni-paderborn.de/pc2.*

[7] **"TotalView Multiprocessor Debugger User's Guide",** Dolphin Toolworks, Version 3.8.0, 1998. *http://www.dolphinics.com/tw.*

[8] **"TotalView Multiprocessor Debugger Installation Guide"**, Dolphin Toolworks, Version 3.8.0, 1998. *http://www.dolphinics.com/tw.*

[9] "**VAMPIRtrace for Solarisx86/ScaMPI Installation and User's Guide",** Pallas GmbH, Release 1.0 for VAMPIRtrace version 1.5, 1998. *http://www.pallas.de.*

[10] **"Review of Performance Analysis Tools for MPI Parallel Programs"**, *http://www.cs.utk.edu/~browne/perftools-review/*.

[11] High Performance Debugger Forum, *http://www.ptools.org/hpdf/.*

[12] NHSE, National HPCC Software Exchange - Parallel Tools Library, *http://www.nhse.org/ptlib.*

[13] TFCC, IEEE CS Task Force on Clustered Computing, - Parallel Tools Library, *http://www.dgs.monash.edu.au/~rajkumar/tfcc.*

[14] The MPICH implementation home page, - *http://www.mcs.anl.gov/mpi/mpich/index.html*.

[15] The Extreme Linux Organisation, *http://www.extremelinux.org.*

[16] **"The Scali parallel tools environment"**, Draft 1999, Scali AS, *http://www.scali.com.*

Appendix A Install ScaMPI

This appendix explains some details of the ScaMPI installation process. Scali software installation onto a Scali system with many cluster nodes is most easily done from the GUI Scali Desktop, see the Scali system guide.

Contents summary:
If you are only using ScaMPI, you can safely skip section A-1 about installation. Section A-2 contains an overview of the file system layout of Scali software under /opt/scali. A-3 give references to useful 3party parallel software.

A-1 Installing

A-1.1 Requirements

Before you can run an MPI program with ScaMPI the following requirements must be satisfied

- Access to a cluster of SCI interconnected nodes with supported SCI drivers installed, or access to a single node.
- The network file system must give a single file system image from all nodes. This restriction regards only the path to programs and files used when executing an MPI program.
- The supported gcc compiler must be installed.
- A license daemon *scald* is installed on (preferably) a front-end of the cluster. The proper license file is obtained from Scali at support@scali.com.
- After installing ScaMPI you get an mpi daemon *mpid* on each node of the cluster. It is important to install the licenses properly **before** you install ScaMPI. Otherwise you have to manually start mpid on each node of the cluster, after installing the license software.
- Finally you can compile and link your program and run it with *mpimon*.
- The required software and instructions areavailable on a Scali cdrom or downloadable from
  http://www.scali.com.
- The install instructions given here are appropriate for installation on a single node at a time. Refer to the ScaMPI Installation Guide for support of automatic install to the entire cluster.

A-1.2 Distribution file

ScaMPI is distributed as a single package file, named ScaMPI.os.arch-x.y.z.package

where:

`x.y.z`	is the release number, e.g. 1.0.2,
`os`	is the operating system, e.g. SunOS5, Linux2 or WinNT4,
`arch`	is the architecture, e.g. sparc-u or i86pc,
`package`	is e.g. pkg, rpm or exe depending on the operating system.

A-1.3 Licensing

ScaMPI is licensed using the FLEXlm license manager system. You will need to install the license server software and obtain a valid demo or permanent license in order to run ScaMPI.

If you have not done so already you must also acquire the Scali license server package - ScaLM. This package contains the Scali vendor daemon, FLEXlm end-user utilities and the necessary information to install and run the license server. Note that you will need this package to get the Scali vendor daemon: scald even though you might have a working (v5 or newer) version of FLEXlm installed on your system.

ScaLM can be downloaded from the Scali Web site at:

```
http://www.scali.com
```

Requests for permanent or demo licenses can be made using the E-mail address: sales@scali.com. For technical questions use: support@scali.com.

Install the ScaLM license manager package, preferably on a front-end of the Scali cluster. Add your license.dat file to /opt/scali/license on the chosen license server. This file holds the licenses for all Scali software.
License.dat contains the host name and port number on the SERVER line as e.g.:

```
SERVER scali-front-end ANY 7788
```

Then install the ScaLMnode package. After an installation on the entire cluster, with input on demand *7788@scali-front-end*, each node has a local /opt/scali/etc/ ScaLM.licserver file, containing a reference to the license server. ScaLM.licserver reads,

LM_LICENSE_FILE=port@host; export LM_LICENSE_FILE

with 7788@scali-front-end as port@host setting in this example.

More detailed information about end-user license administration can be found with the ScaLM package in the online HTML version of the “FLEXlm End-User Manual”. To view this online documentation use your web browser to open the file:

file:///opt/scali/license/doc/htmlman/index.html

FLEXlm is a trademark of Globetrotter Software, Inc.

A-1.4 Removing an earlier release of ScaMPI

Only a single release of ScaMPI can be installed at a system. If an earlier release of ScaMPI already is installed at a system this release must be removed before a new release can be installed.

```
Solaris:     # pkgrm ScaMPI
Linux:       # rpm -e ScaMPI
WinNT: Use Windows uninstall to remove the old ScaMPI release.
Start‘Settings‘Control Panel^^Add/Remove Programso, o=ScaMPI, o^Add/Remove.
```

All files and mpid deamon processes will be removed.

A-1.5 Installing a new release

Only a single release of this software can be installed on a system. If an earlier release is already installed on the system please check the remove section A-1.4 before continuing:

```
Solaris:     # pkgadd -d ScaMPI.os.arch-x.y.z.pkg ScaMPI
Linux:       # rpm -ivh ScaMPI.os.arch-x.y.z.rpm
WinNT:       Use Windows install to locate the ScaMPI release or double click on the
ScaMPI executable.
             Start‘Settings‘Control Panel^^Add/Remove Programso,
             o^Install ..., or ^^ScaMPI.os.arch-x.y.z.exe.
```

Normally the ScaMPI package will be installed in /opt/scali (WinNT: c:\opt\scali).

A-1.6 Verification of installation

After an attempted installation of ScaMPI you may check the installation.

```
Solaris:     % pkginfo ScaMPI
Linux:       % rpm -qi ScaMPI
WinNT:       Check if ScaMPI is listed among the installed Windows programs.
             ? Start‘Settings‘Control Panel^^Add/Remove Programso, o=ScaMPI
```

A-2 Scali packages file system layout

Almost all Scali software package files are installed under a common directory /opt/scali. Under this directory files are organised as follows:

- **bin** : All normal executables to be run by users.
- **sbin** : All daemons and executables to be run by administrators.
- **libexec** : Executables which are hidden from normal invokation (e.g. used by applications under bin).
- **lib** : Libraries used by our application and by end users.
- **include** : Include files for libraries under lib.
- **doc** : All documentation.
- **etc** : Configuration files for our software.
- **license** : Files belonging to a 3rd party licensing system (FlexLM)
- **examples** : Example applications, source and documentation
- **contrib** : 3rd party software adapted to Scali software.
- **init.d** : boot/startup scripts are installed here.
  Soft links are made to system startup catalogues (rc.d and init.d under /etc)

A-3 Useful 3rd party parallel software

It is possible to integrate any kind of parallel MPI software with ScaMPI. Contact sales@scali.com for latest information on availability or how to get your parallel software option.
Other good places for information about 3rd party parallel software are [12, 13, 14, 15 and 16].

List of figures

3-1 TotalView main window 26
3-2 TotalView MPI process window 27
3-3 VAMPIR snapshots 30
3-4 VAMPIR profiled MPI program call tree window 31
4-1 Startup phase 1 36
4-2 Startup phase 2 37
4-3 Startup phase 3 38
4-4 Channel 40
4-5 Eagerbuffer 41
4-6 Transporter 42
4-7 Thresholds for different communication protocols 43
4-8 Inlining 43
4-9 Eagerbuffering 44
4-10 Transporter 45

List of tables

1-1 Abbreviations....8
1-2 Basic terms....8
1-3 Typographic conventions....9
3-1 Environment variables on Unix....16
3-2 Environment variables for Unix on Windows NT....17
3-3 Basic options to mpimon....20
3-4 Mpimon parameters....21
3-5 Numeric input....21
3-6 Complete list of mpimon options....22
3-7 Environment setup for TotalView....28
3-8 Setup for VAMPIR and VAMPIRtrace....31
4-1 Libraries....35
4-2 Startup phase 1....36
4-3 Startup phase 2....37
4-4 Startup phase 3....38
4-5 Stopping steps.....39
5-1 Namespace pollution....48
5-2 Trouble shooting....50

Index

C
CCS resource management software 57
compiler flags 18
compiler support 17
compiling 12, 17
D
debugging 25
E
error messages 49
G
gdb 28
I
install 59
install requirements 59
L
libraries 35
linking 19
M
MPI test programs 13
mpiboot 35
mpid 35
mpimon 35
mpisubmon 35
P
profiling 29
R
running 20
S
Scali Desktop GUI 20
setup 16, 17
T
Totalview 25
TotalView Users's Guide 57
trouble shooting 50
V
Vampir 29
VAMPIRtrace Installation and User's Guide 57